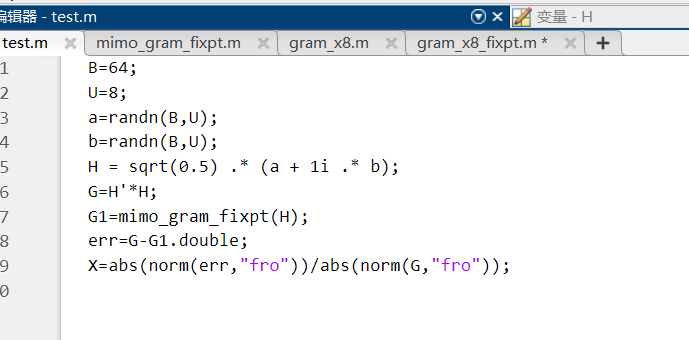
**面向大规模 MIMO 检测的矩阵乘法设计**

组长 吴非 519021910924

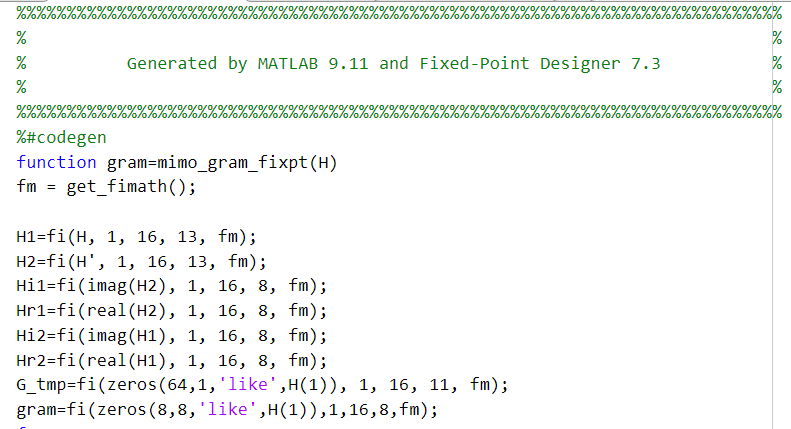
组员 贾鑫鹏 5190121911317

1. 软件Gram矩阵乘法设计

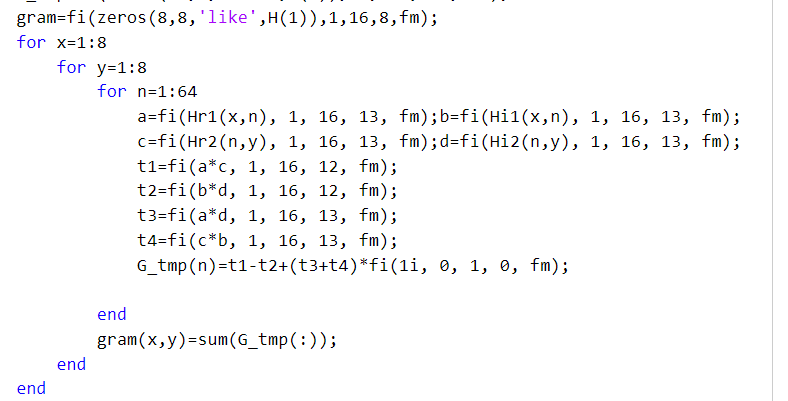
1.代码



这部分代码由实验指导中给出是浮点型的Gram矩阵算法，并且列出了误差的计算。



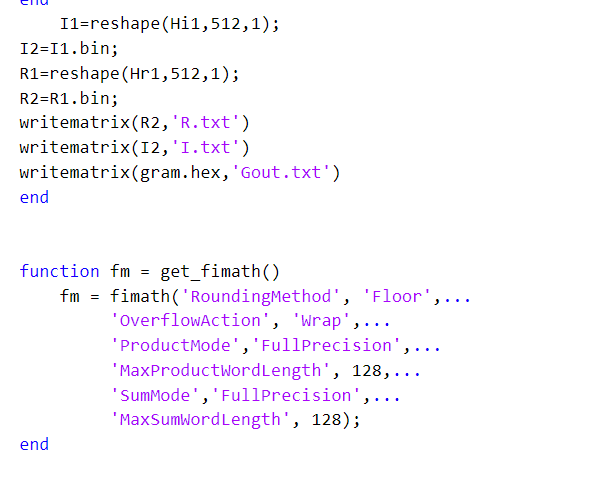
该部分是对计算需要的值进行定义和提取这是定点化后的代码。



矩阵的计算过程，在硬件结构中我们采用的是(a+bi)\*(c+di)=(ac+bd)-(ad+bc)i的计算公式，所以在计算过程中产生的中间变量不多，分别列出并提取，用工具箱对其进行量化,主要以fi函数对数据进行量化，例fi(x,1,16,8),即对x变量进行定点量化，有符号数，16位字长，8位是小数。量化后分别得到了G1，H1，在做差值，得到顶点量化后的误差X1，通过多次的运行得到结果.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X1 | 0.0117 | 0.0010006 | 0.0084 | 0.0079 | 0.0595 | 0.0073 |

可以看出误差基本稳定在6%以下符合要求。



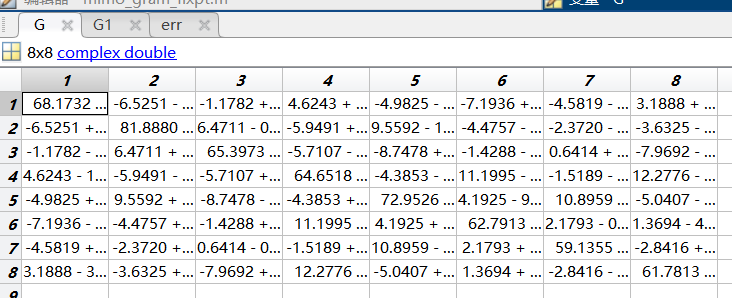
在将需要的H1矩阵的虚步实部输出作为硬件的输入，Gout作为相应参考的结果数据。

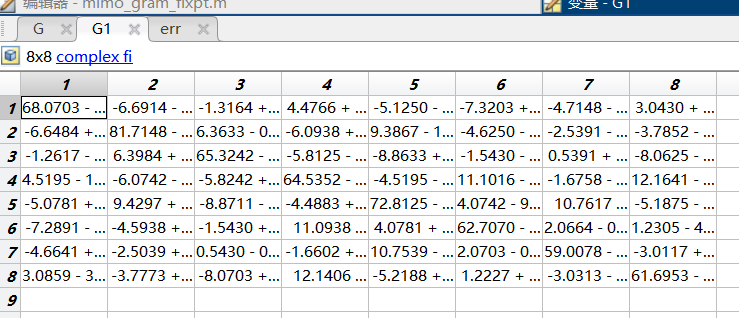
2.定点量化的考虑

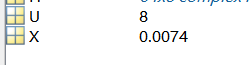
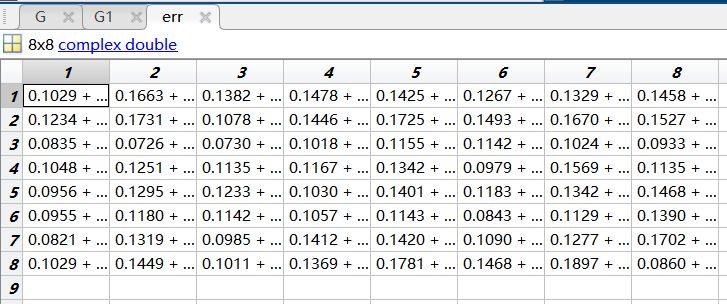
对于输出矩阵G而言，在定点量化的过程中，首先在浮点型运行了几次代码，得到G矩阵，其中的结果不会超过200，而将整数位设置为8位完全满足了整数部分的需求，而小数部分8位的精度也使得误差不会超过6%，并且整数小数各8位的模式也方便了硬件部分的输入输出及结果整理。

对于输入数据H的量化，也采用各自8位的方式，整数部分会超出预期的性能，但是在硬件中会方便实现，并在且在后续的调试过程中更容易发现问题。

软件的计算过程为了保证精度均采用了13位小数的方式，但后面为了验证与硬件电路的适配性而改为了8位。







单次软件运行结果及误差截图，实部虚部是随机生成具，数值上平等，通过对实部的观察可以看出，软件的仿真符合逻辑且运行正确。

1. 硬件Gram矩阵乘法设计
2. 基本架构设计及数据的流动

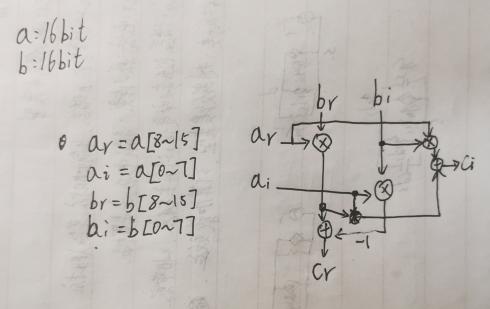
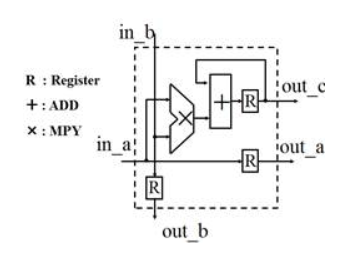
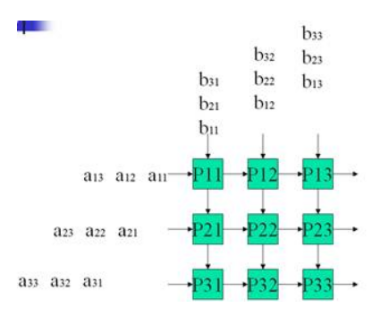
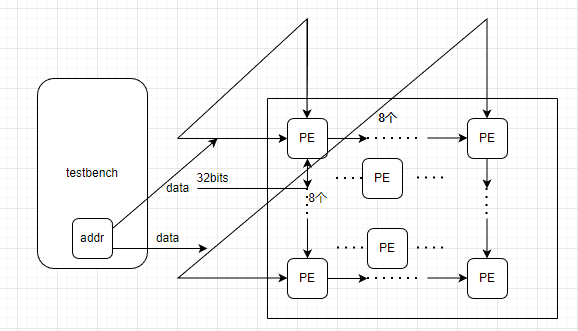


图1 pe\_array基本结构 图2 PE单元结构 图3 PE中乘法器结构

整体图：



31:16bits：实部 15：0虚部

以(3,3)\*(3,3)的矩阵计算为例，相应的基本计算结构如上图，右侧为左图中对应的PE单元的具体架构设计，以PE11为例，前矩阵的第一行与后矩阵中的第一列依次流入并相乘，在通过其中的寄存器进行累加，则的到的结果为PE11=a11\*b11+a12\*b21+a13\*b31，此时PE11的结果为最终结果G11的值，对应的本次实验的矩阵算法为（8\*64）\*（64\*8）的矩阵乘法，则需要8\*8的PE基本单元，每个PE需要流入64\*2个基本数据。

在PE的乘法器部分，是一个单独的32位的乘法计算器，将输入的32bit的a，b两个数据分为16bit的4部分，由公式(a+b\*i)\*(c+d\*i)=ac-bd+(ad+bc)i得到最终计算结果，再通过公式ac-bd=a(c-d)+d(a-b)与ad+bc=b(c+d)+d(a-b)优化得到对应得硬件结果。

最终64个PE单元的输出是最终的结果。

1. 模块接口和功能的主要描述

对于PE部分，输入的数据为每个周期进入的两个32bit数据，输出由三部分组成，两个个是对当前输入数据进行一个时钟的延迟后传递给后面的PE，另一个输出是在达到某一周期后所得到的最终结果。

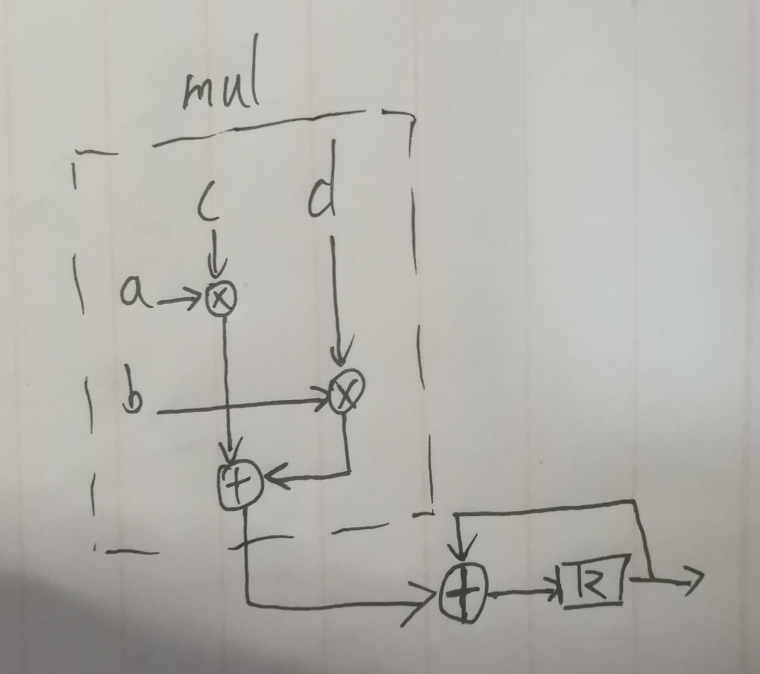
对于pe\_array，输入数据为按照顺序输入的H矩阵中的元素，同时拥有16个输入端口，在输出方面，共有64个输出端口，即PE单元的数量。

delay文件用来产生部分使能信号，来负责控制电路需要的时序。

而testbench文件中，将已有的输入数据读取到memory中，同时产生地址（地址的产生较为简单，所以在其内部生成），输出相应的数据输入进模块内。

乘法器部分由4个基本的乘法器组成，如上图所示，以两个矩阵中的两个元素为输入，将其分实部，虚部共4个部分，其中高16位是实部，低十六位是虚部，在经过四个乘法器与两个加法器的计算得到两者之积。

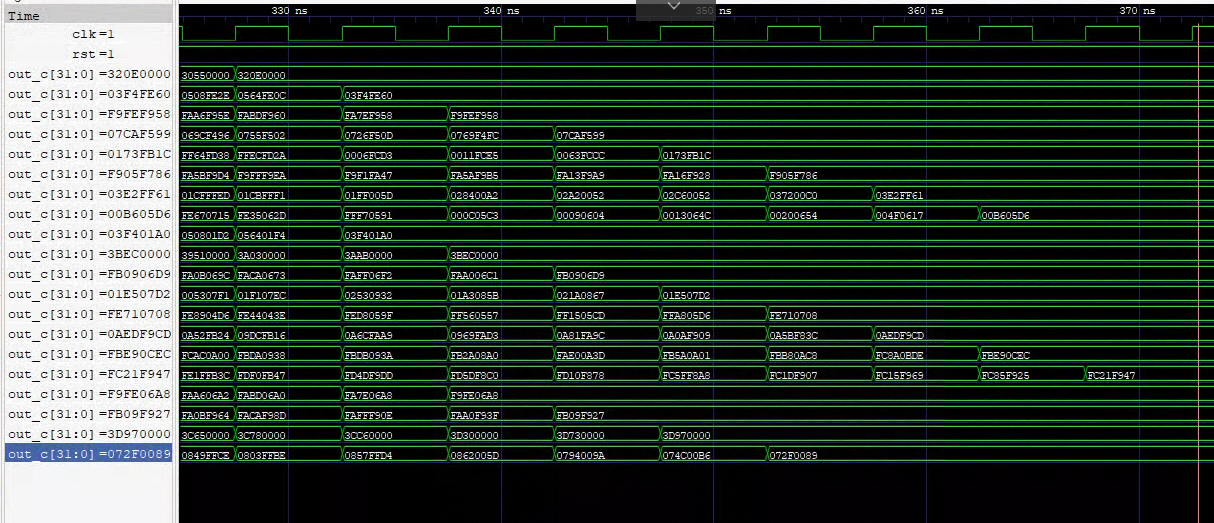
1. 关键路径

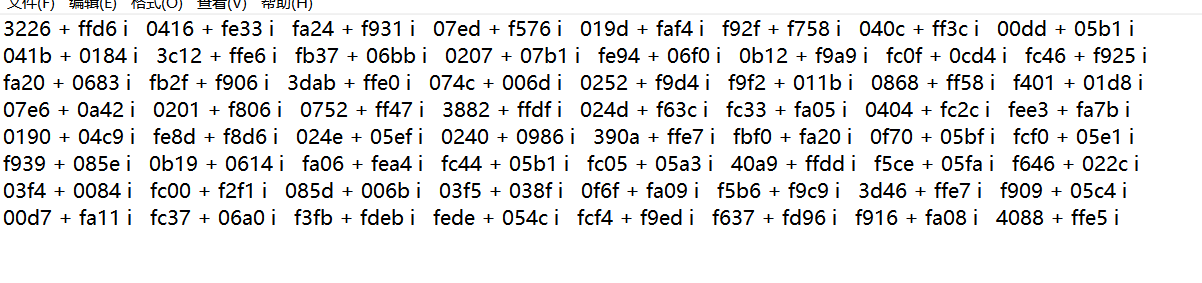
在此次的架构中主要的关键路径来自于乘法器部分，即单个的PE单元中，在PE之间都存在寄存器相互连接，所以只能在其内部，我们的算法中在乘法器部分会存在这四个乘法器并行，两个加法器并行，从而产生了一个加法器一个乘法器的路径，而在乘法器外部，数据仍要与上一个周期的数据进行累加操作，所以外部存在一个加法器，所以关键路径为两个加法器与一个乘法器。

1. 结果与时序图

、

输出的开始截图满足架构的时序。

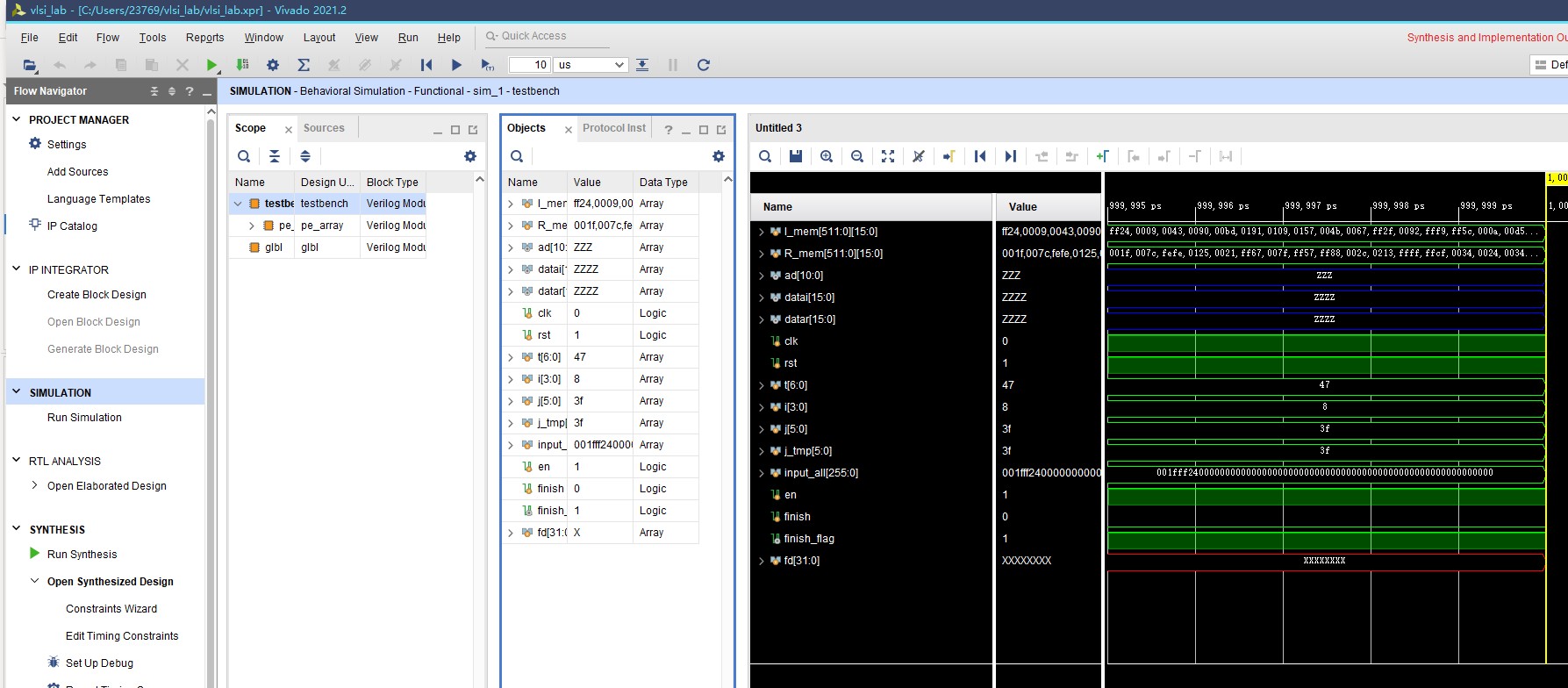




输出的波形以及部分最终得到的数值，按顺序与软件所得的数据进行对比，结果存在差距，这在后面的部分会进行说明，整体的数值与理论结果相近，可以说明架构的正确性，通过观察与数据的追踪来看，两者的误差存在于实部或虚部的低五位里面，即误差范围在0.125以内。

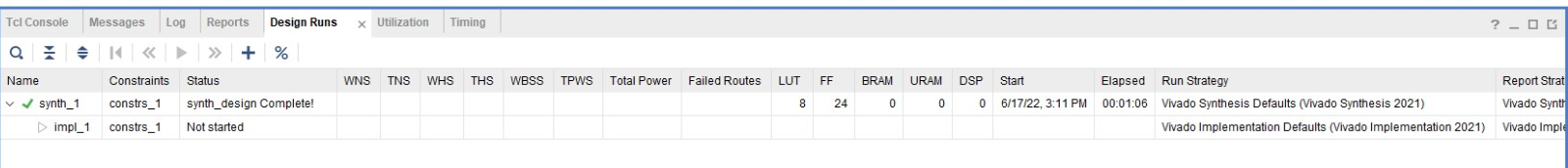
1. 综合结果

波形：

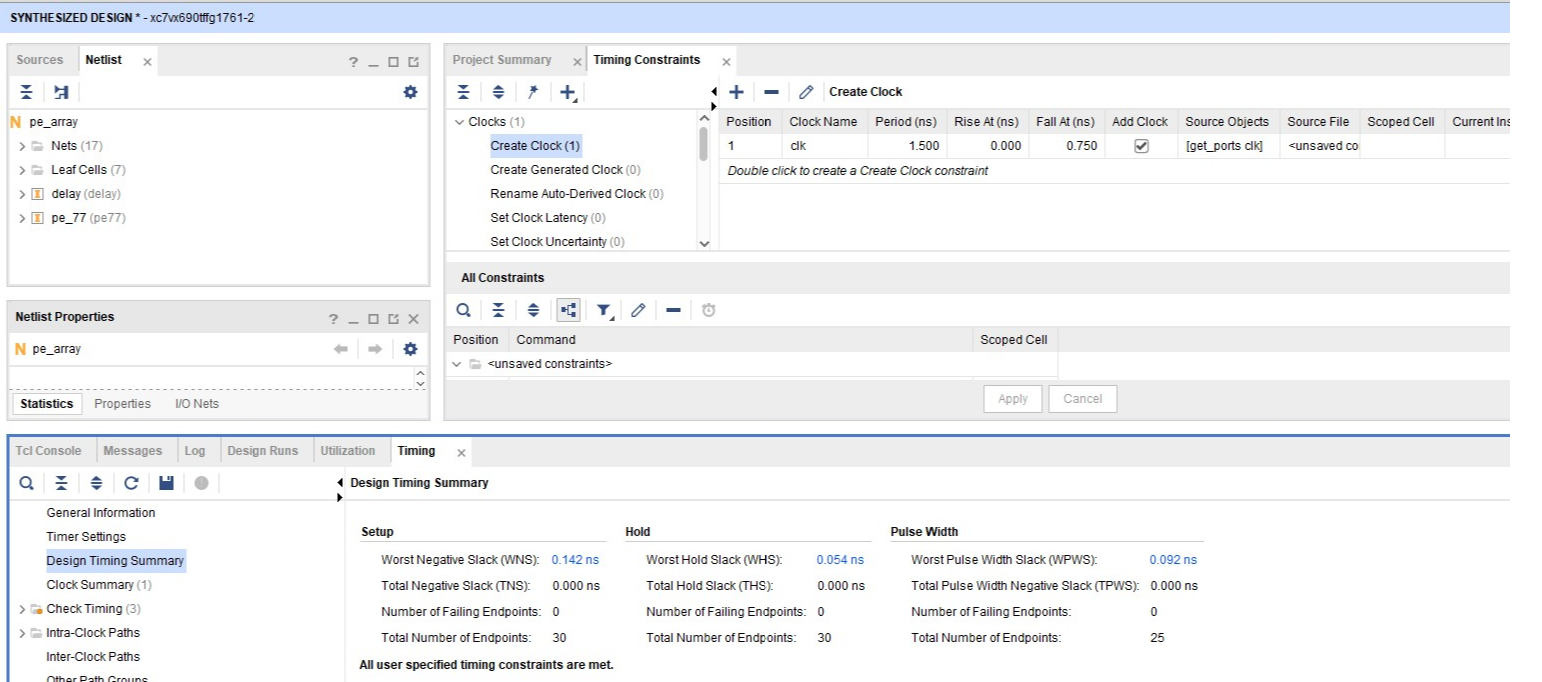


Synthesis:

资源：

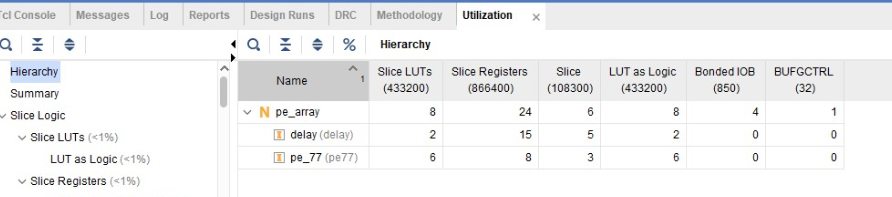


时序：

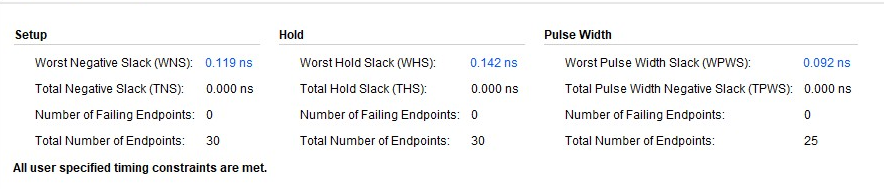


Implement：

资源：



时序：



1. 对截图中得到的数据进行提取得到表格

面积1：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| name | Slice LUTs(433200) | Slice Registers(866400) | Slice(108300) | LUTs as Logic(433200) | Bonded IOB(850) | BUFGCTRL(32) |
| pe\_array | 8 | 24 | 6 | 8 | 4 | 1 |
| delay | 2 | 15 | 5 | 2 | 0 | 0 |
| pe\_77 | 6 | 8 | 3 | 6 | 0 | 0 |

时钟频率：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Position | Clockname | Period(ns) | RiseAt(ns) | FallAt(ns) |
| 1 | clk | 1.5 | 0 | 0.75 |

吞吐率：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Contraints | status | LUT | FF | URAM | BRAM | DSP | Elapsed |
| Synth\_1 | Contrs\_1 | synth\_design Complete | 8 | 24 | 0 | 0 | 0 | 0:01:06 |

1. 性能的计算

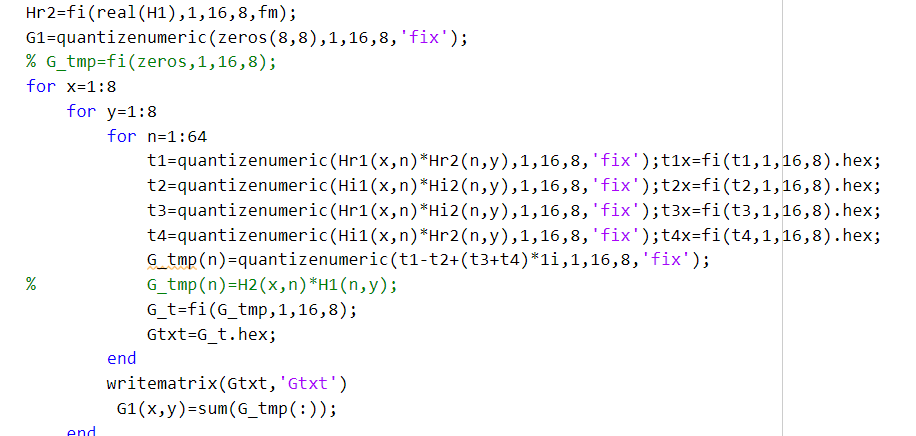
计数单位的计数周期为71(64+7)，即需要71个时钟周期来计算一个Gram矩阵，而从综合结果来看，结构可以满足clk=1.5ns，则fc=660MHz所以计算得吞吐率为

Throughput=(1/71)\*fclk=9.3e+6

进一步可以得到硬件效率

Hardware Efficiency=Throughput/(LUTs+FFs+DSP\*280)=2.9e+5

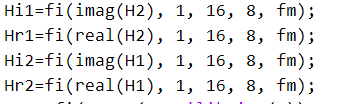
1. 遇到的问题
2. 在一开始的运行中，最终结果与实际得到的Gram矩阵始终存在着巨大差异，在经过排查后发现，复数矩阵的转置与常规矩阵的转置并不相同，在经过修改后得到了和实际矩阵结果接近的正确结果。
3. 在实际的运行过程中，硬件结果与软件结果始终无法完全匹配，在小数部分会存在低六位bit存在误差，通过逐步对比与分析，得出结论在matlab中的计算存在一定问题。在经过了大量的运行和多种修改后仍未得到想要结果，下面是在软硬件相匹配的过程中进行的尝试。



在最开始的软件尝试中，我们采用了quantizer与quantize的组合来进行进一步的量化，但没有得到想要的结果，之后又尝试了quantizenumeric函数，但得到的结果与fi函数所给出的结果并无太大差距。

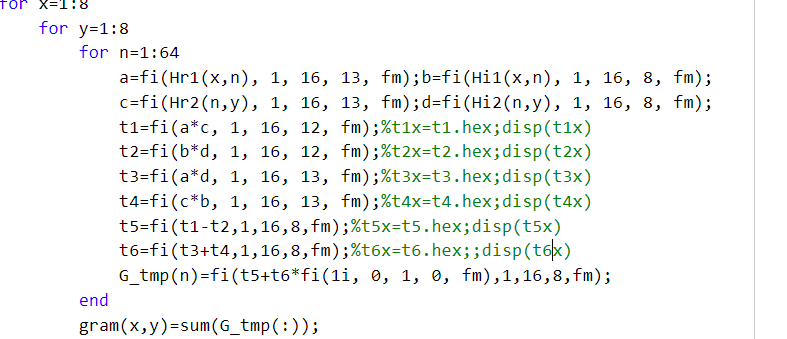
第二次我们尝试令过程更加细化，将每一步的计算都采用定点的形式，将数据产生多份，在一定程度上增加了接近率，但没有得到想要的结果。

第三次尝试改变精度，首先在软件上通过对精度的改变来获得不同组的数据，

即改变了图中“8”的大小。

但当数据的小数位大于11或者小于6时都会产生数据的溢出，所以该方法并不可通。

在verilog中，我们尝试将硬件中间过程的算法从原来的32位扩展到64位，来观察最终与软件的差异，最终结果虽然有所接近但是仍存在误差，



在最后我们通过追踪过程中的值进行逐步对比校验以及人工计算，发现了产生误差的部分，但是却没有办法去更改这一状况，在排查中发现



这两部中的结果与硬件中的结果存在一定差异，在计算过程中，每次的结果与硬件相比都会在最低位的部分相差1，-1或者相同，但在64次的累加后可能会在低6位的部分产生误差。

1. 分工

吴非：

架构的设计和架构图的绘制

verilog硬件部分代码的编写

和贾同学讨论(debug)并修改代码。

用python生成了测试输入向量(1，1+i等)。

vivado的综合

贾鑫鹏：

软件代码的编写与修改

参与Verilog的调试

编写实验报告。